PROJECT DEVELOPMENT MANUAL

TEAM-592538

AN AUTOMATED PREDICTION MODEL FOR DIABETIC RETINOPATHY USING CNN

The perpetration stage of any design is a true display of the defining moments that make a design a success or a failure. The installation and operationalization of the system or system variations in a product terrain is appertained to as the perpetration step. After the system has been tried out and approved by the stoner, the phase is started. This phase continues until the system is operating in product in agreement with the defined stoner conditions.

Language / Technology Used

Python is the language used for developing the detection and process of the system and for image processing using Convolutional Neural Network.

Libraries / Algorithms Used

NUMPY

NumPy is a python library used for working with arrays. It also has functions for working in the sphere of direct algebra, fourier transfigure, and matrices. NumPy was created in 2005 by Travis Oliphant. It's an open source design and you can use it freely. NumPy stands for Numerical Python. In Python we've lists that serve the purpose of arrays, but they're slow to reuse. NumPy aims to give an array object that's over to 50x faster than traditional Python lists. At the core of the NumPy package, is the array object. This encapsulates n- dimensional arrays of homogeneous data types, with numerous operations being performed in collected law for performance. There are several important differences between NumPy arrays and the standard Python sequences

SCIKIT LEARN

Scikit- learn(Sklearn) is the most useful and robust library for machine literacy in Python. It provides a selection of effective tools for machine literacy and statistical modeling including bracket, retrogression, clustering and dimensionality reduction via a harmonious interface in Python. This library, which is largely written in Python, is erected upon NumPy, SciPy and Matplotlib.

It was firstly called scikits learn and was originally developed by David Cournapeau as a Google summer of law design in 2007. latterly, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, from FIRCA(French Institute for Research in Computer Science and robotization), took this design at another position and made the first public release(v0.1 beta) on 1stFeb. 2010.

TENSORFLOW

TensorFlow is a Python library for fast numerical computing. It was created and is maintained by Google and released under the Apache2.0 open source license. It's a foundation library that can be used to produce Deep literacy models directly or by using wrapper libraries that simplify the process erected on top of Tensor Flow.

calculation in TensorFlow is described in terms of data inflow and operations in the structure of a directed graph.

- Bumps perform calculation and have zero or further inputs and labors. Data that moves between bumps are known as tensors, which are multi-dimensional arrays of real values.

- Edges The graph defines the inflow of data, branching, looping and updates to state. Special edges can be used to attend geste within the graph, for illustration staying for calculation on a number of inputs to complete.

- Operation An operation is a named abstract calculation which can take input attributes and produce affair attributes. For illustration, you could define an add or multiply operation.

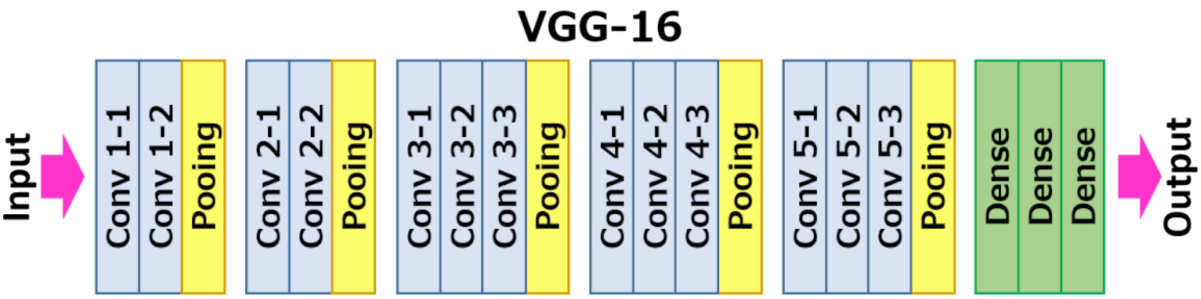
KERAS

Keras is one of the most popular Python libraries for Deep literacy. Keras is simple, flexible and important. It's a library that provides you colorful tools to deal with neural network models. These tools enable you to fantasize and understand the model. These represent the factual neural network model. These models group layers into objects. There are two types of Models available in Keras The successional model and the Functional model.

Keras Sequential Model is simple and easy to use model. It's a direct mound of styles that groups a direct mound of layers into atf.keras.Model. According to its name, its main task is to arrange the layers of the Keras in successional order. In this model, the data inflow from one subcaste to another subcaste. The inflow of data is continued until the data reaches the final subcaste. utmost of the Neural Networks use the successional API Model

VGG16

VGG16 is a simple and extensively used Convolutional Neural Network(CNN) Architecture. The VGG16 Architecture was developed and introduced by Karen Simonyan and Andrew Zisserman from the University of Oxford. VGG16 is used in numerous deep literacy image bracket ways and is popular due to its ease of perpetration. It was one of the notorious model with large kernel- sized pollutants(11 and 5 in the first and alternate convolutional subcaste, independently) with multiple 3 × 3 kernel- sized pollutants one after another.



Configurations:

The ConvNet configurations are outlined in figure **. The nets are appertained to their names(A-E). All configurations follow the general design present in armature and differ only in the depth from 11 weight layers in the network A(8 conv. and 3 FC layers) to 19 weight layers in the network E(16 conv. and 3 FC layers). The range of conv. layers the number of channels) is rather small, starting from 64 in the first subcaste and also adding by a factor of 2 after each maximum-pooling subcaste, until it reaches 512.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Sample Code:

IMPORTING DEPENDENCIES:

```
[2] from google.colab import drive
    drive.mount('/content/drive/')

    Mounted at /content/drive/

[3] !pip install -q keras

    !pip install tensorflow

    Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.14.0)
    Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
    Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
    Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
    Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
    Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
    Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
    Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
    Requirement already satisfied: ml-dtypes==0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
    Requirement already satisfied: numpy>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.23.5)
    Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
    Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
    Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
    Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
    Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
    Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
    Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5.0)
    Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
    Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.34.0)
    Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.59.2)
    Requirement already satisfied: tensorboard<2.15,>=2.14 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.1)
    Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
    Requirement already satisfied: keras<2.15,>=2.14.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
    Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.41.3)
    Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow) (2.17.3)
    Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow) (1.0.0)
    Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow) (3.5.1)
    Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow) (2.31.0)
    Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow) (0.7.2)
    Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow) (3.0.1)
    Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (5.3.2)
    Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (0.3.0)
    Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (4.9)
    Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow) (1.3.1)
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (3.3.2)
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (3.4)
    Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (2.0.7)
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (2023.7.22)
    Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.15,>=2.14->tensorflow) (2.1.3)
    Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (0.5.0)
    Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow) (3.2.2)

[5] import numpy as np
    import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Activation, Dense, Flatten
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.metrics import categorical_crossentropy
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.layers import BatchNormalization
    from tensorflow.keras.layers import Conv2D, MaxPooling2D
    from matplotlib import pyplot as plt
    from sklearn.metrics import confusion_matrix
    import itertools
    import matplotlib.pyplot as plt
    %matplotlib inline
```

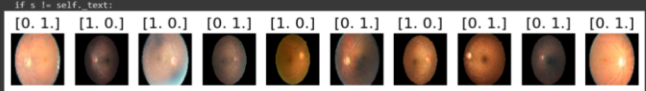
Data Collection and analysis:

Setting up the file paths of the training, and validation data.Creating batches of data using the ImageDataGenerator

```
[6] test_path ='/content/drive/MyDrive/Colab_Notebooks/ml_project/test'
    train_path = '/content/drive/MyDrive/Colab_Notebooks/ml_project/train'
    valid_path = '/content/drive/MyDrive/Colab_Notebooks/ml_project/valid'

    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    test_batches = ImageDataGenerator().flow_from_directory(test_path, target_size=(224, 224), classes=['dr', 'nodr'], batch_size=10)
    train_batches = ImageDataGenerator().flow_from_directory(train_path, target_size=(224, 224), classes=['dr', 'nodr'], batch_size=10)
    valid_batches = ImageDataGenerator().flow_from_directory(valid_path, target_size=(224, 224), classes=['dr', 'nodr'], batch_size=10)

    Found 40 images belonging to 2 classes.
    Found 40 images belonging to 2 classes.
    Found 40 images belonging to 2 classes.
```

Defining a function named plots for plotting images with their corresponding label

```
[8] def plots(ims, figsize=(12,6), rows=1, interp=False, titles=None):
        if type(ims[0]) is np.ndarray:
            ims = np.array(ims).astype(np.uint8)
            if(ims.shape[-1] != 3):
                ims = ims.transpose((0,2,3,1))
        f = plt.figure(figsize=figsize)
        cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
        for i in range(len(ims)):
            sp = f.add_subplot(rows, cols, i+1)
            sp.axis('Off')
            if titles is not None:
                sp.set_title(titles[i], fontsize=16)
            plt.imshow(ims[i], interpolation=None if interp else 'none')

[9] imgs , labels = next(train_batches)

    plots(imgs, titles=labels)

    /usr/local/lib/python3.10/dist-packages/matplotlib/text.py:1279: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
      if s != self._text:
```



```
[11] from tensorflow import keras
     from tensorflow.keras.applications import VGG16

     # Create a VGG16 model
     vgg16_model = VGG16()

     Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
     553467096/553467096 [==============================] - 6s 0us/step
```

Loading the pre-Trained VGG16 model provided by keras

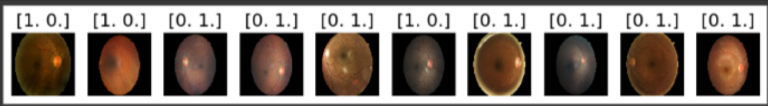
```
[12] vgg16_model = keras.applications.vgg16.VGG16()

vgg16_model.summary()

Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 1000)              4097000

=================================================================
Total params: 138357544 (527.79 MB)
Trainable params: 138357544 (527.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
[14] type(vgg16_model)

keras.src.engine.functional.Functional

[15] from tensorflow.keras.models import Sequential  # Import the Sequential class

     model = Sequential()  # Create a Sequential model

     for layer in vgg16_model.layers[:-1]:
         model.add(layer)  # Add layers to the model

model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

=================================================================
Total params: 134260544 (512.16 MB)
Trainable params: 134260544 (512.16 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling the model

Fitting the model on the training data and validating on the validation data.

```
[21] model.fit(train_batches, steps_per_epoch=4,
                    validation_data=valid_batches, validation_steps=4, epochs=5, verbose=2)

Epoch 1/5
4/4 - 79s - loss: 0.9520 - accuracy: 0.4500 - val_loss: 0.8186 - val_accuracy: 0.4500 - 79s/epoch - 20s/step
Epoch 2/5
4/4 - 64s - loss: 0.8458 - accuracy: 0.3750 - val_loss: 0.8482 - val_accuracy: 0.4750 - 64s/epoch - 16s/step
Epoch 3/5
4/4 - 65s - loss: 0.8421 - accuracy: 0.4750 - val_loss: 0.8448 - val_accuracy: 0.4750 - 65s/epoch - 16s/step
Epoch 4/5
4/4 - 46s - loss: 0.7756 - accuracy: 0.4750 - val_loss: 0.8078 - val_accuracy: 0.4250 - 46s/epoch - 12s/step
Epoch 5/5
4/4 - 65s - loss: 0.6977 - accuracy: 0.6500 - val_loss: 0.8087 - val_accuracy: 0.4250 - 65s/epoch - 16s/step
<keras.src.callbacks.History at 0x7a29decd17e0>

[22]  test_imgs, test_labels = next(test_batches)
plots(test_imgs, titles=test_labels)
```



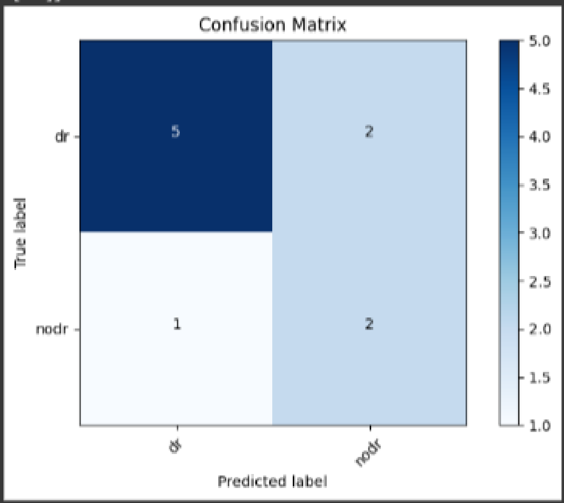

```
[23] test_labels = test_labels[:,0]
test_labels

array([1., 1., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)

import matplotlib.pyplot as plt
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
```

Loading the testing data and making predictions using the predict method of the model. Plotting the confusion matrix of the model's predictions.

```
[26] from sklearn.metrics import confusion_matrix  # Import the confusion_matrix function

     cm = confusion_matrix(test_labels, np.round(predictions[:, 0]))
```

```
[27] import itertools  # Import the itertools module


     cm_plot_labels = ['dr', 'nodr']
     plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
```

```
Confusion matrix, without normalization
[[5 2]
 [1 2]]
```




```
from sklearn.metrics import accuracy_score

# Assuming you have the ground truth labels in test_labels and model predictions in predictions
accuracy = accuracy_score(test_labels, np.round(predictions[:, 0]))
print("Accuracy Score:", accuracy)
```

```
Accuracy Score: 0.7
```

```
[43] from sklearn.metrics import classification_report

     # Assuming you have the ground truth labels in test_labels and model predictions in predictions
     class_report = classification_report(test_labels, np.round(predictions[:, 0]), target_names=cm_plot_labels)
     print("Classification Report:")
     print(class_report)
```

```
Classification Report:
              precision    recall  f1-score   support

          dr       0.83      0.71      0.77         7
        nodr       0.50      0.67      0.57         3

    accuracy                           0.70        10
   macro avg       0.67      0.69      0.67        10
weighted avg       0.73      0.70      0.71        10
```

```
[28] model.save('diabetic_retinopathy.h5')

     /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
       saving_api.save_model(
```

Defining a function named fix_layer0 for fixing the batch input shape and data type of the model. Defining a function named get_model for loading the pre-trained model.

Defining a function named preprocess_image for preprocessing an image before making a prediction.

```
def fix_layer0(filename, batch_input_shape, dtype):
    with h5py.File(filename, 'r+') as f:
        model_config = json.loads(f.attrs['model_config'])
        layer0 = model_config['config']['layers'][0]['config']
        layer0['batch_input_shape'] = batch_input_shape
        layer0['dtype'] = dtype
        f.attrs['model_config'] = json.dumps(model_config).encode('utf-8')

def get_model():
    global model,graph
    model = tf.keras.models.load_model('diabetic_retinopathy.h5', compile=False)
    print(" * Model Loaded!!")
    graph = tf.get_default_graph()

def preprocess_image(image, target_size):
    if image.mode != "RGB":
        image = image.convert("RGB")
    image = image.resize(target_size)
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)

    return image

print(" * Loading keras model.....")
fix_layer0('diabetic_retinopathy.h5', [None, 224, 224,3], 'float32')
get_model()
```

```
 * Loading keras model.....
 * Model Loaded!!
```

RESULTS

```python
import tensorflow as tf

# assuming `prediction` is a symbolic tensor
image = Image.open('/content/drive/MyDrive/Colab_Notebooks/ml_project/test/dr/54_right.jpeg')
image.show()
processed_image = preprocess_image(image, target_size=(224, 224))
prediction = model.predict_on_batch(processed_image)
prediction_shape = tf.shape(prediction)
num_samples = prediction_shape[0]

def condition(tempDr, tempndr):
    return tempDr > tempndr

def true_fn(tempDr, tempndr):
    return 1;

def false_fn(tempDr, tempndr):
    return 0;

with tf.compat.v1.Session() as sess:
    for i in range(sess.run(num_samples)):
        tempDr = prediction[i][0]
        tempndr = prediction[i][1]
        result = true_fn(tempDr, tempndr)
        output = tf.cond(condition(tempDr, tempndr), lambda: true_fn(tempDr, tempndr), lambda: false_fn(tempDr, tempndr))
        if result==1:
          print('Diabetic Retinopathy')
        else:
          print('No Diabetic Retinopathy')


image.show()
```

```
Diabetic Retinopathy
```

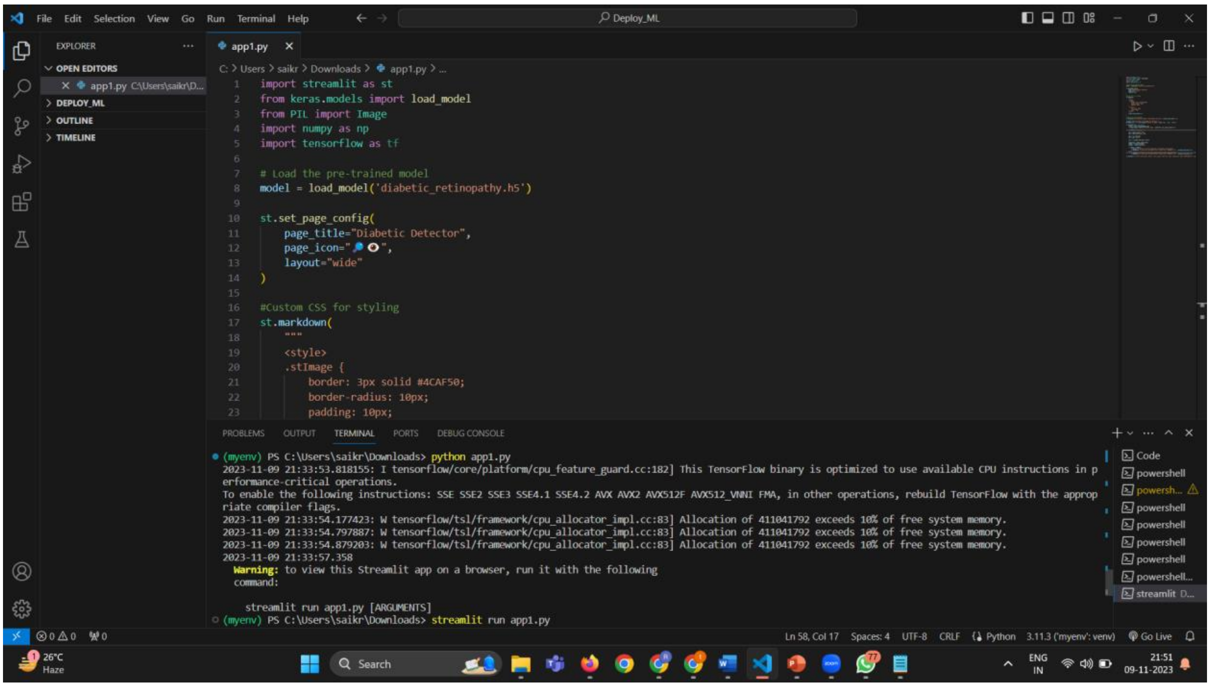
```python
import tensorflow as tf

# assuming `prediction` is a symbolic tensor
image = Image.open('/content/drive/MyDrive/Colab_Notebooks/ml_project/test/nodr/46_right.jpeg')
image.show()
processed_image = preprocess_image(image, target_size=(224, 224))
prediction = model.predict_on_batch(processed_image)
prediction_shape = tf.shape(prediction)
num_samples = prediction_shape[0]

def condition(tempDr, tempndr):
    return tempDr > tempndr

def true_fn(tempDr, tempndr):
    return 1;

def false_fn(tempDr, tempndr):
    return 0;

with tf.compat.v1.Session() as sess:
    for i in range(sess.run(num_samples)):
        tempDr = prediction[i][0]
        tempndr = prediction[i][1]
        result = false_fn(tempDr, tempndr)
        output = tf.cond(condition(tempDr, tempndr), lambda: true_fn(tempDr, tempndr), lambda: false_fn(tempDr, tempndr))
        if result==1:
          print('Diabetic Retinopathy')
        else:
          print('No Diabetic Retinopathy')
```

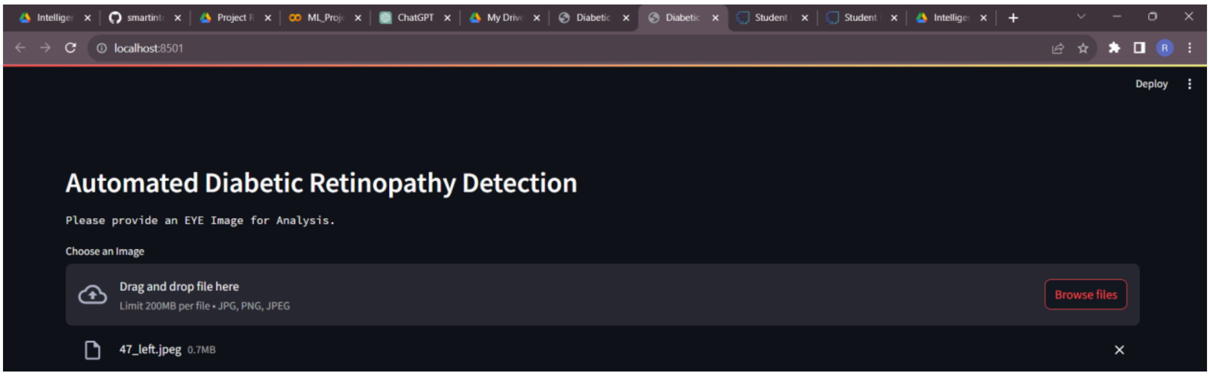
```
No Diabetic Retinopathy
```

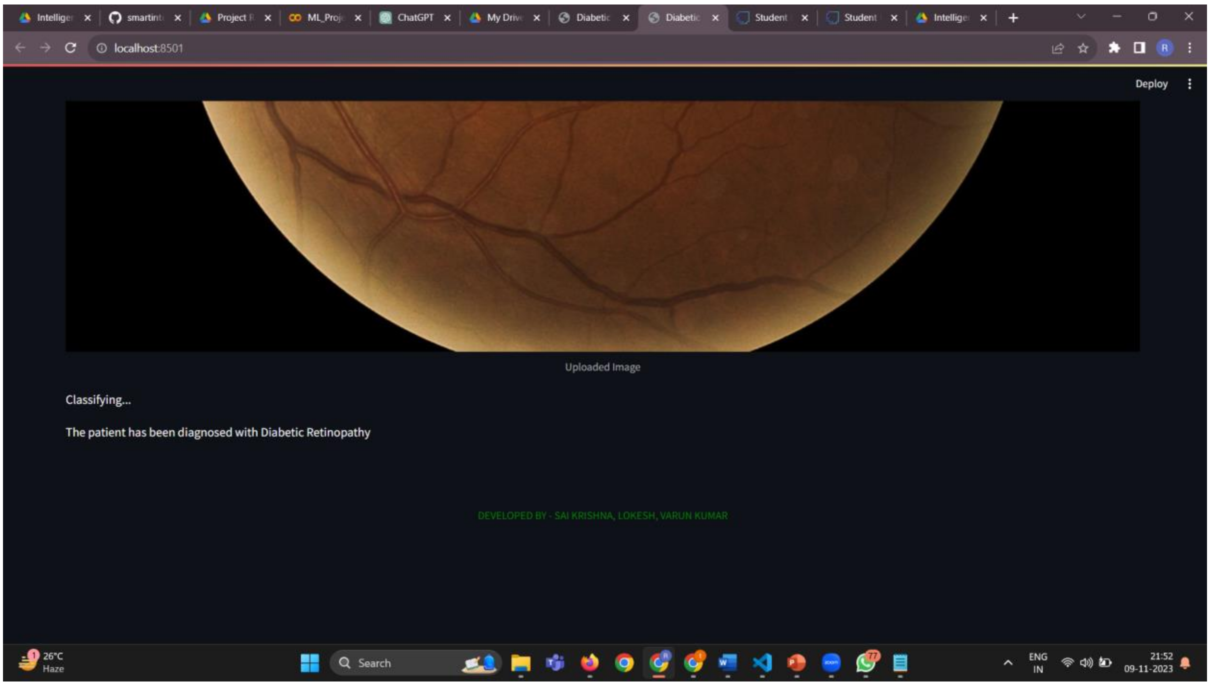
DEPLOYMENT:

In the final phase of the project, the developed automated prediction model for Diabetic Retinopathy, powered by Convolutional Neural Networks (CNN), was ready for deployment. To make this invaluable tool accessible to healthcare professionals and patients alike, we opted for the user-friendly and interactive platform offered by Streamlit. Streamlit provided an efficient and elegant means of deploying the model, allowing users to upload retinal images effortlessly and receive real-time predictions.

The model, saved in the .h5 format, was seamlessly integrated into the Streamlit application. This format, known for its compatibility with deep learning models, ensured that the model's architecture, weights, and training history were preserved during deployment. Streamlit's simplicity and flexibility enabled the creation of an intuitive user interface, where users could easily upload their retinal images, receive immediate predictions, and access interpretability tools to understand the model's decisions.







CODES AND PROJECT DOCUMENTS:

https://drive.google.com/drive/folders/1bc9w8_qH_vAwrQRAweHIJ1s8ig4WtV2h?usp=drive_link